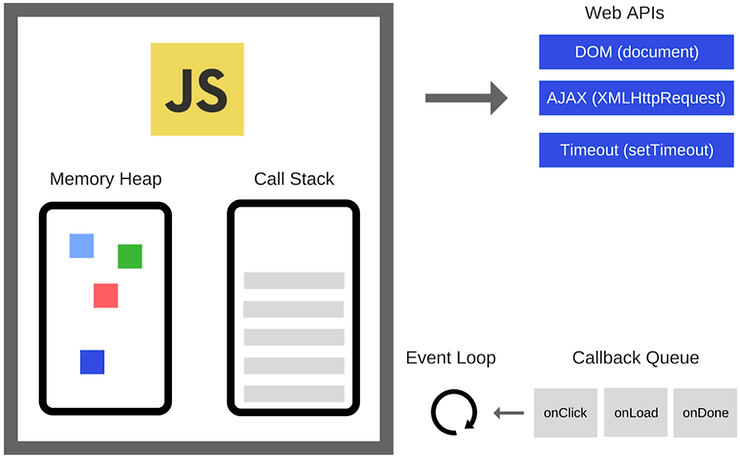
Node JS

Asnchronous Node.js:

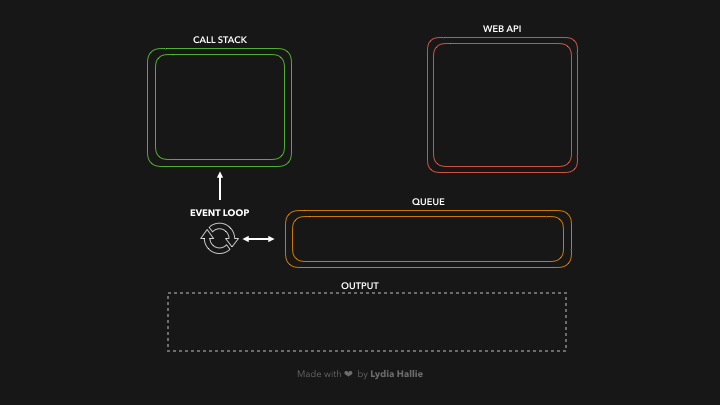


Note: Asynchronous functions will run and execute only after all the synchronous functions have been executed. Until then, they will be processed in the background.

When the program starts executing, it first gets wrapped inside an anonymous main() function.

This is automatically defined by NodeJS. So main() gets pushed first to the callback stack.

* **Memory Heap** — Objects are allocated in a heap which is just a name to denote a large mostly unstructured region of memory.
* **Call Stack** — This represents the single thread provided for JavaScript code execution. Function calls form a stack of frames. It is responsible for keeping track of all the operations in line to be executed. Whenever a function is finished, it is popped from the stack. It is a **LIFO queue** (Last In, First Out).
* **Browser or Web APIs** — They are built into your web browser and are able to expose data from the browser and surrounding computer environment and do useful complex things with it. They are not part of the JavaScript language itself, rather they are built on top of the core JavaScript language, providing you with extra superpowers to use in your JavaScript code. For example, the Geolocation API provides some simple JavaScript constructs for retrieving location data so you can say, plot your location on a Google Map. In the background, the browser is actually using some complex lower-level code (e.g. C++) to communicate with the device’s GPS hardware (or whatever is available to determine position data), retrieve position data, and return it to the browser environment to use in your code. But again, this complexity is abstracted away from you by the API.
* **Event or Callback Queue** — It is responsible for sending new functions to the track for processing. It follows the queue data structure to maintain the correct sequence in which all operations should be sent for execution.



Microtask and macrotask Queue

In JavaScript, there are two types of task queues: the microtask queue and the macrotask queue. These task queues are used by the event loop to execute code in the correct order.

The microtask queue is a queue that is processed before the macrotask queue. Microtasks are generally short-lived and executed immediately after the current task has completed. Examples of microtasks include Promises and the **process.nextTick** method in Node.js. When a microtask is added to the queue, it is executed as soon as the current task has completed.

The macrotask queue is a queue that is processed after the microtask queue. Macrotasks are typically longer-lived and represent bigger units of work, such as I/O operations, timers, and user events. Examples of macrotasks include **setTimeout**, **setInterval**, and I/O operations. When a macrotask is added to the queue, it is executed only after the microtask queue has been emptied.

The event loop processes the microtask queue before the macrotask queue in order to prioritize short and important tasks. This ensures that code that depends on microtasks, such as Promise callbacks, is executed as soon as possible.

Microtask

Example: process.nextTick, Promises, queueMicrotask, MutationObserver

macrotask Queue

**Examples:** setTimeout, setInterval, setImmediate, requestAnimationFrame, I/O, UI Rendering

https://www.thetechplatform.com/post/the-javascript-event-loop-explained

https://blog.logrocket.com/5-ways-to-make-http-requests-in-node-js/#:~:text=Currently%2C%20Got%20is%20the%20most,19%20million%20downloads%20per%20week.&text=Below%20is%20a%20quick%20example,('got')%3B%20got.

https://www.twilio.com/blog/2017/08/http-requests-in-node-js.html

**7 ways to make HTTP requests in Node.js**

There are multiple ways to make HTTP requests in Node.js. We can do so using the standard HTTP/HTTPS module, of course, or we could use one of a number of npm packages that make our lives much easier.

### 1. Standard Node.js HTTP(S) module

Node.js comes with both [HTTP](https://nodejs.org/api/http.html) and [HTTPS](https://nodejs.org/api/https.html) modules in the standard library.

2. Request

npm init

npm install request

3. [Axios](https://blog.logrocket.com/how-to-make-http-requests-like-a-pro-with-axios/) is a very popular promise-based request library. It is an HTTP client available for both browser and Node.js. It also includes handy features like intercepting request and response data, and the ability to automatically transform request and response data to JSON.

We can install Axios with the following command:

npm install --save axios

### 4. SuperAgent

Similarly to Axios, [SuperAgent](https://github.com/visionmedia/superagent) is another popular library primarily used for making AJAX requests in the browser but works in Node.js as well. Install SuperAgent with the following command:

npm install superagent@6.1.0

What is cool about SuperAgent is that you have other useful functions that you can chain onto requests such as query() to add parameters to the request. We’ve been just manually adding them in the URL in the previous examples, but notice how SuperAgent gives you a function to do this:

Just like with Axios you don’t have to parse the JSON response yourself, which is pretty cool.

5. [Got](https://www.npmjs.com/package/got) is another popular HTTP request library for Node.js. It claims to be a “human-friendly and powerful HTTP request library for Node.js.” It also features a promise-based API, and HTTP/2 support and its pagination API are Got’s USPs. Currently, Got is the most popular HTTP client library for Node.js, with over 19 million downloads per week.

We can install Got with the command below:

npm install --save got

This code example is pretty similar to Axios, but with two main differences:

1. We needed to pass {responseType: 'json'} as the second parameter to indicate that the response was in JSON format.
2. The status code header was called statusCode, not status.

### node-fetch

[node-fetch](https://www.npmjs.com/package/node-fetch) is another hugely popular HTTP request library for Node.js — in the first week of December 2020, it was downloaded more than 20 million time, as per [npm trends](https://www.npmtrends.com/node-fetch).

In their own words, “node-fetch is a lightweight module that brings the Fetch API (window.fetch) to Node.js.” Its features include consistency with the browser-based window.fetch and native promise and async functions.

We can install node-fetch with the command below:

npm install --save node-fetch

1. fetch did not need an explicit GET method; the HTTP verb can be sent as a method key in the second parameter, which is an object. For instance: {method: 'GET'}
2. Another difference is that the header is an object with a get method to get header values. We called res.headers.get('date') to get the value of the date response header
3. The final difference was the need to unwrap a promise to get the body as JSON with await res.json(). It seemed like a bit of extra work, but that is how the browser Fetch API [response](https://developer.mozilla.org/en-US/docs/Web/API/Body/json) works.

## Error handling techniques

https://stackify.com/node-js-error-handling/

To handle the errors effectively, we need to understand the error delivery techniques.

There are four fundamental strategies to report errors in Node.js:

1. try…catch blocks
2. Callbacks
3. Promises
4. Event emitters

### try…catch blocks

### Callbacks

A [callback function](https://blog.logrocket.com/a-complete-guide-to-the-node-js-event-loop/" \t "_blank) (commonly used for asynchronous code) is an argument to the function in which we implement error handling.

The purpose of a callback function is to check the errors before the result of the primary function is used. The callback is usually the final argument to the primary function, and it executes when an error or outcome of the operation emerges.

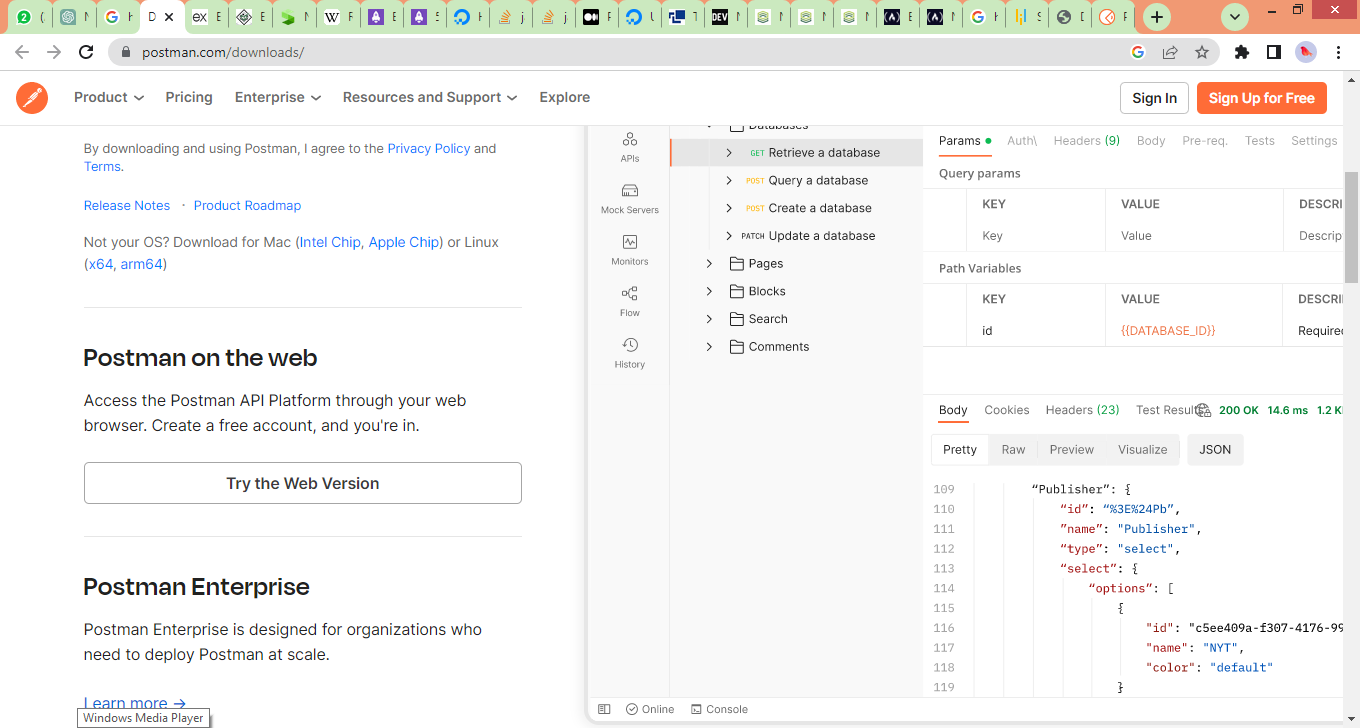
Here’s the syntax for a callback function:

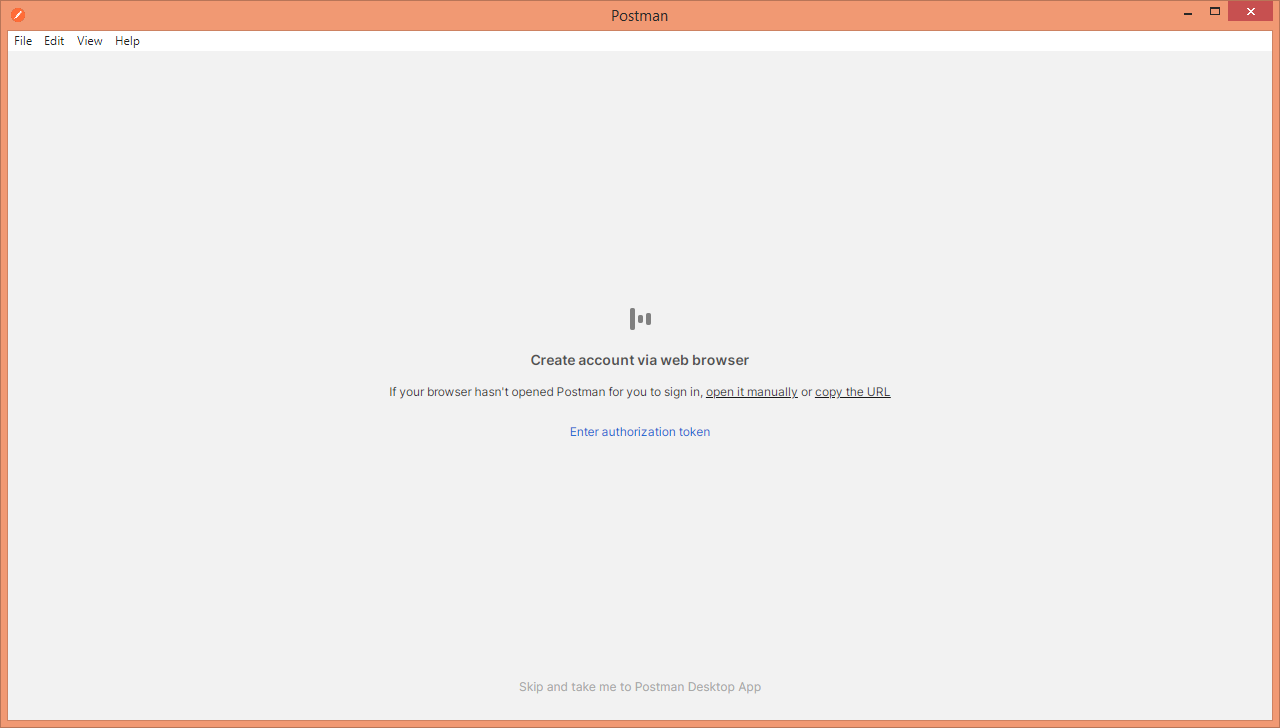
function (err, result) {}

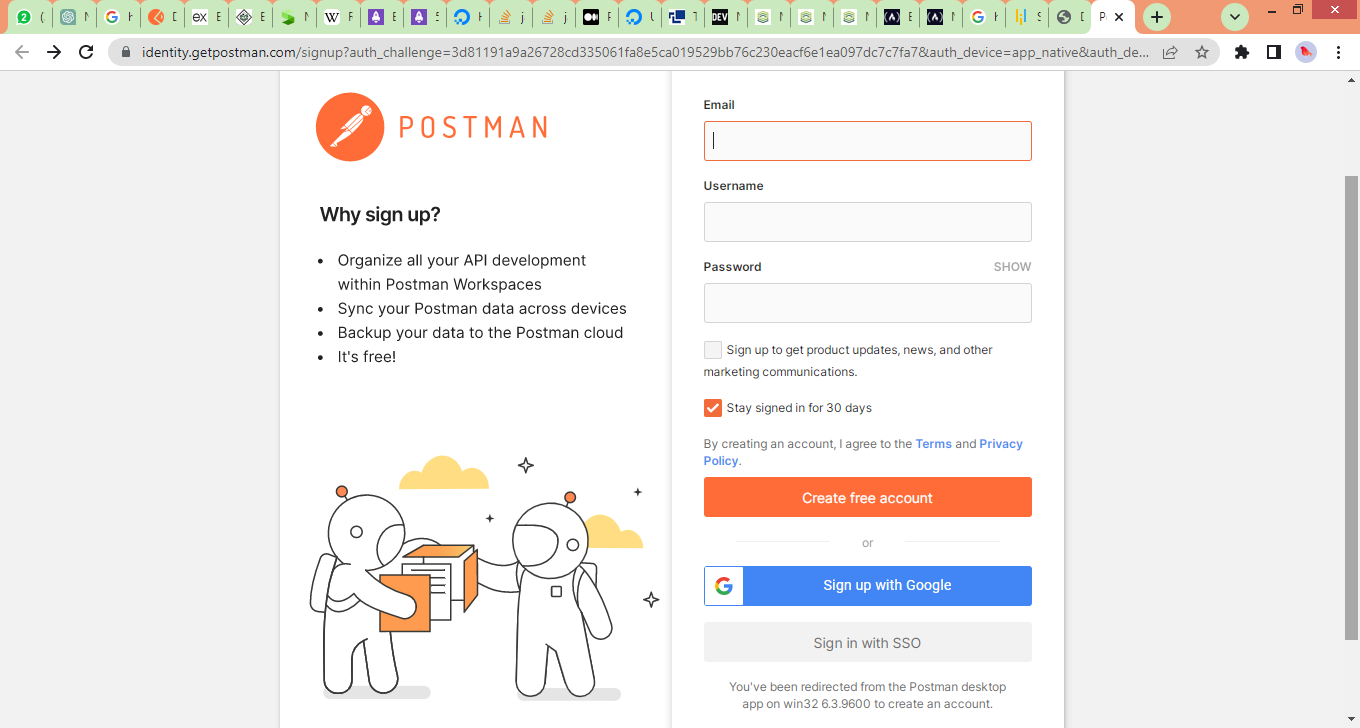
CURL

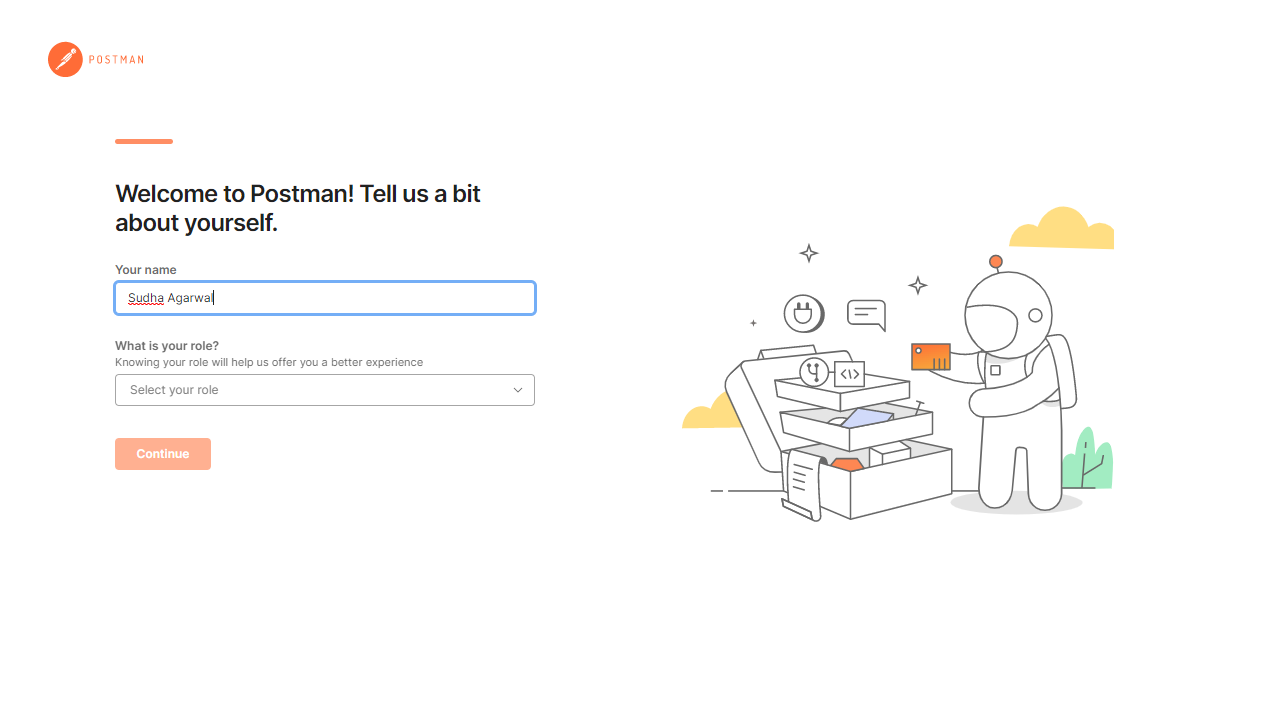
In a separate terminal window, we’ll communicate with the server using [cURL](https://github.com/curl/curl), a CLI tool to transfer data to and from a network.

Postman









Deployment to Heroku from Github

npm install express

npm install nodemon

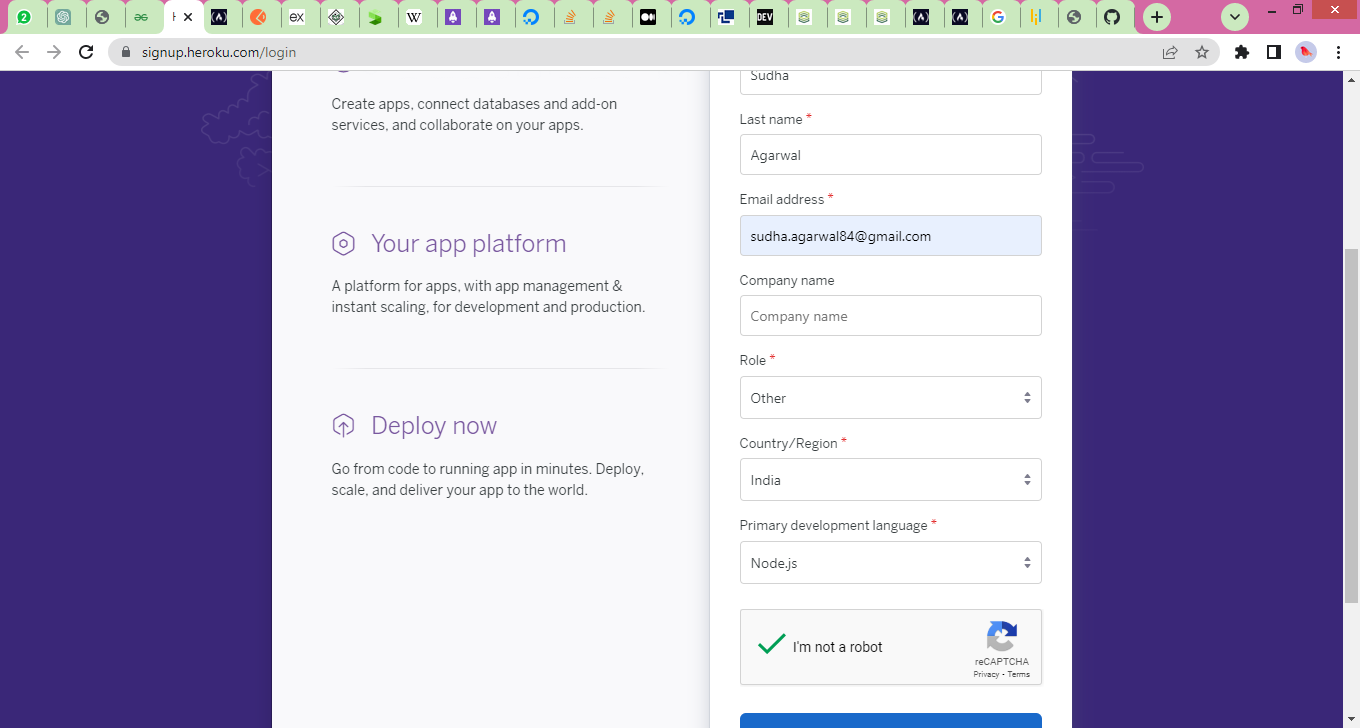
"start": "node app.js",

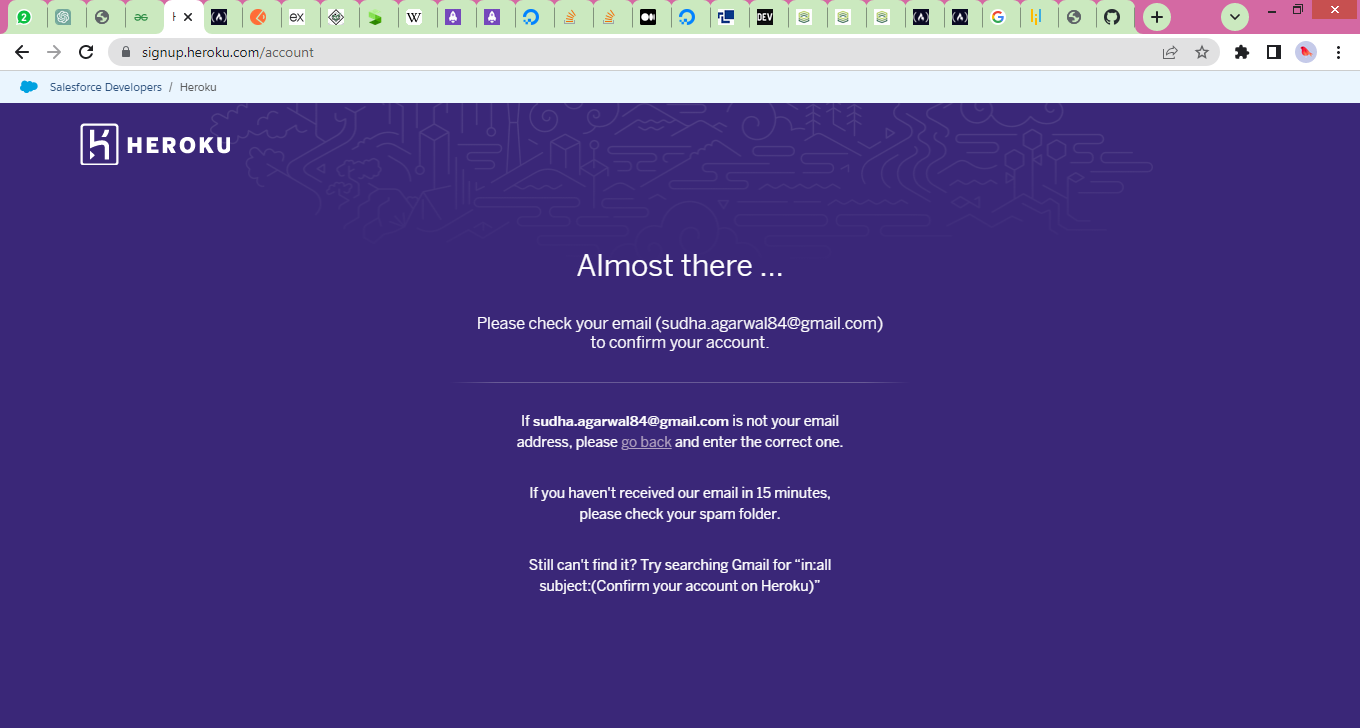
"dev": "nodemon app.js"

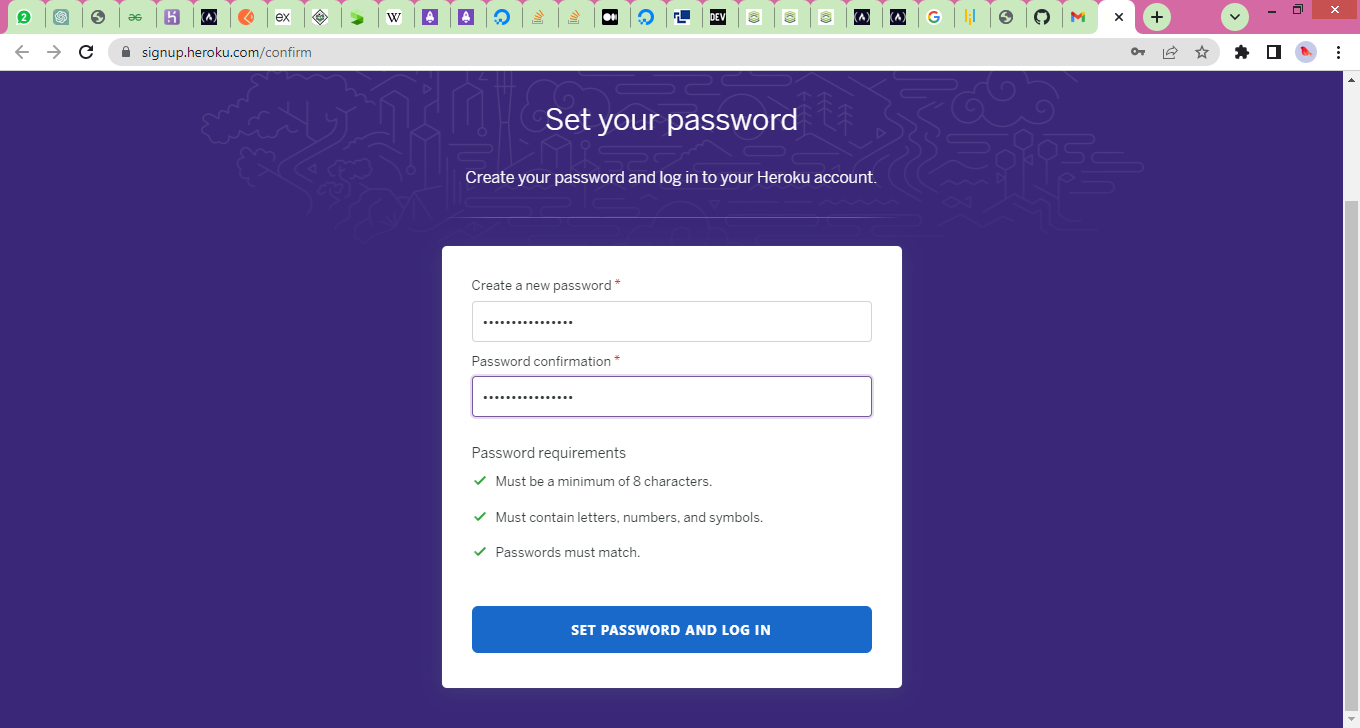
npm run dev - to start the application

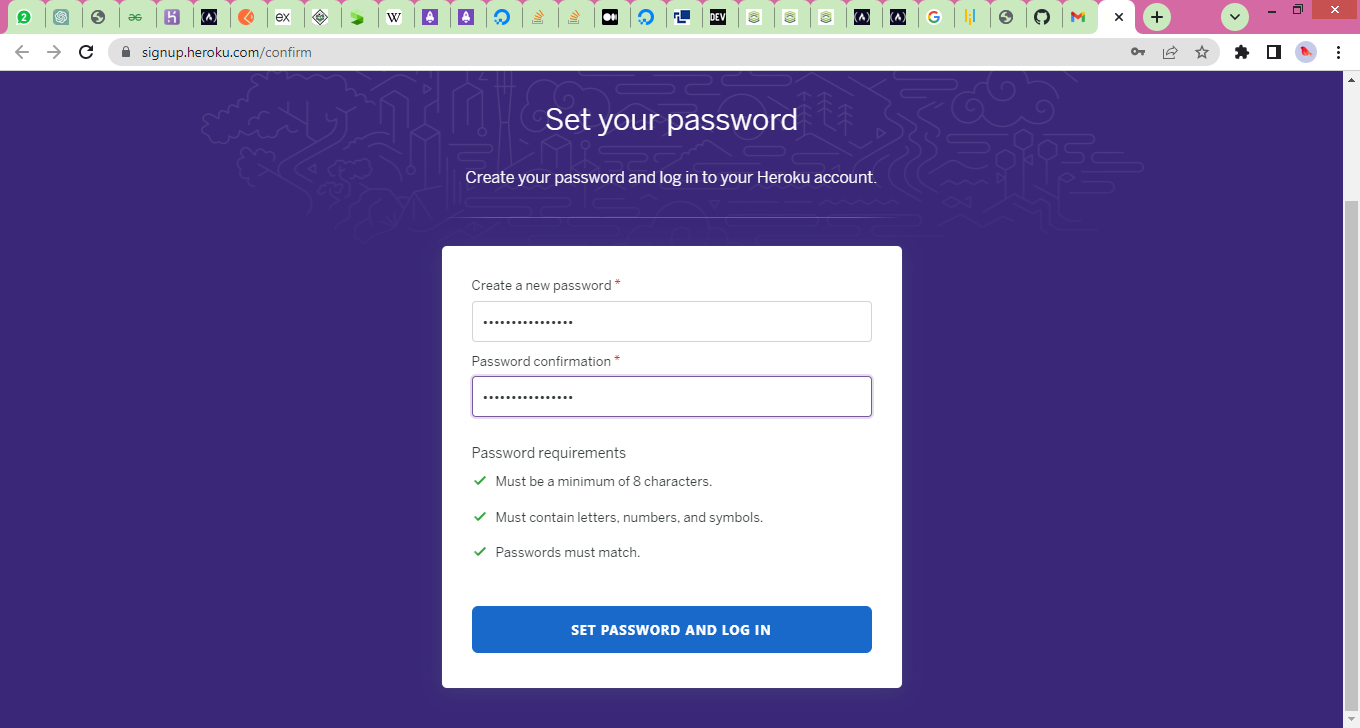
**Step 4: Deploying the Node.js app to Heroku**

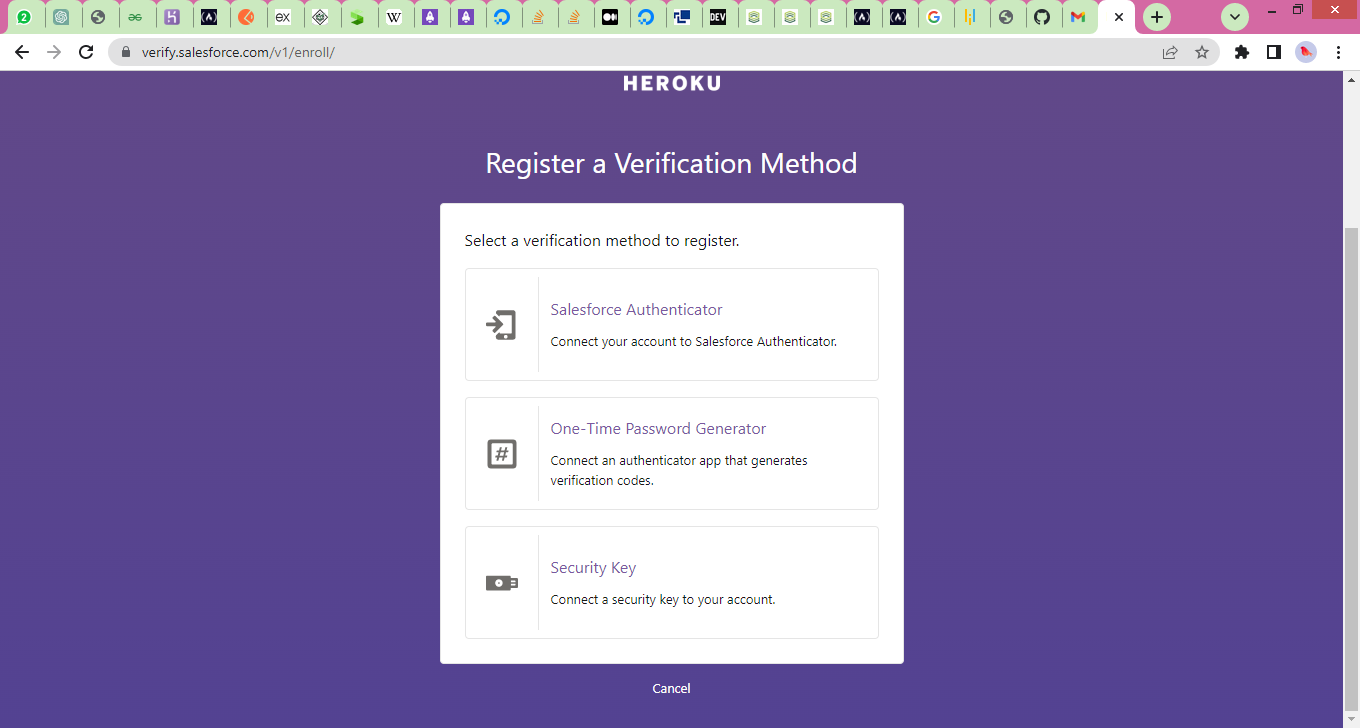
* Create a free account on[Heroku](https://signup.heroku.com/login)
* https://signup.heroku.com/login
* After completing all processes now login to your account.
* Click on the “**Create new app**”.
* Open the Deploy tab and scroll to the “Deployment method” section of the tab.
* Select GitHub as a method. It will show a **“Connect to GitHub”** option where we add provide our GitHub repository. If you are doing it for the first time, Heroku will ask permission to access your GitHub account you have to permit that.Here, you can search for your **GitHub** repository and click**connect**for creating a connection.
* After that, the Deployment section will show up where you can select pick them up and deploy or Manual Deployment, click**Enable Automatic Deploys.**





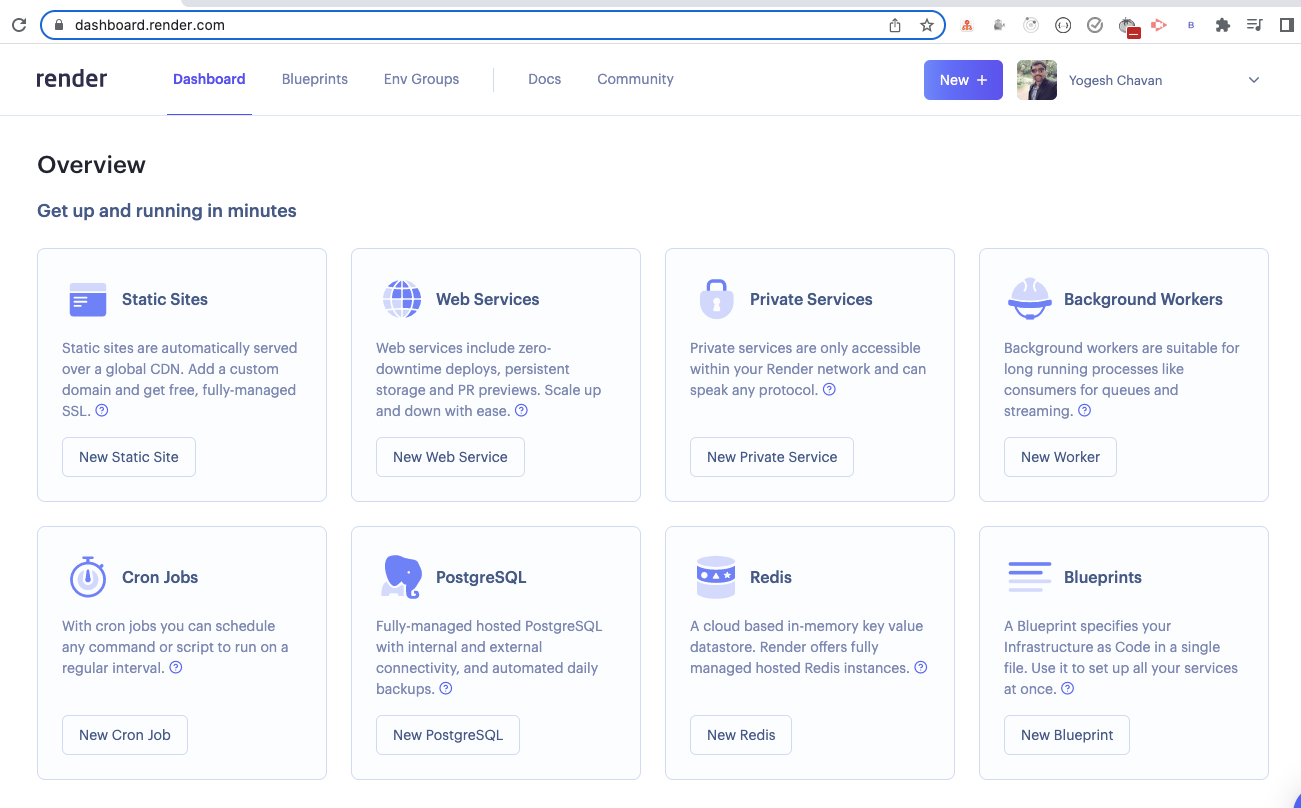






Render

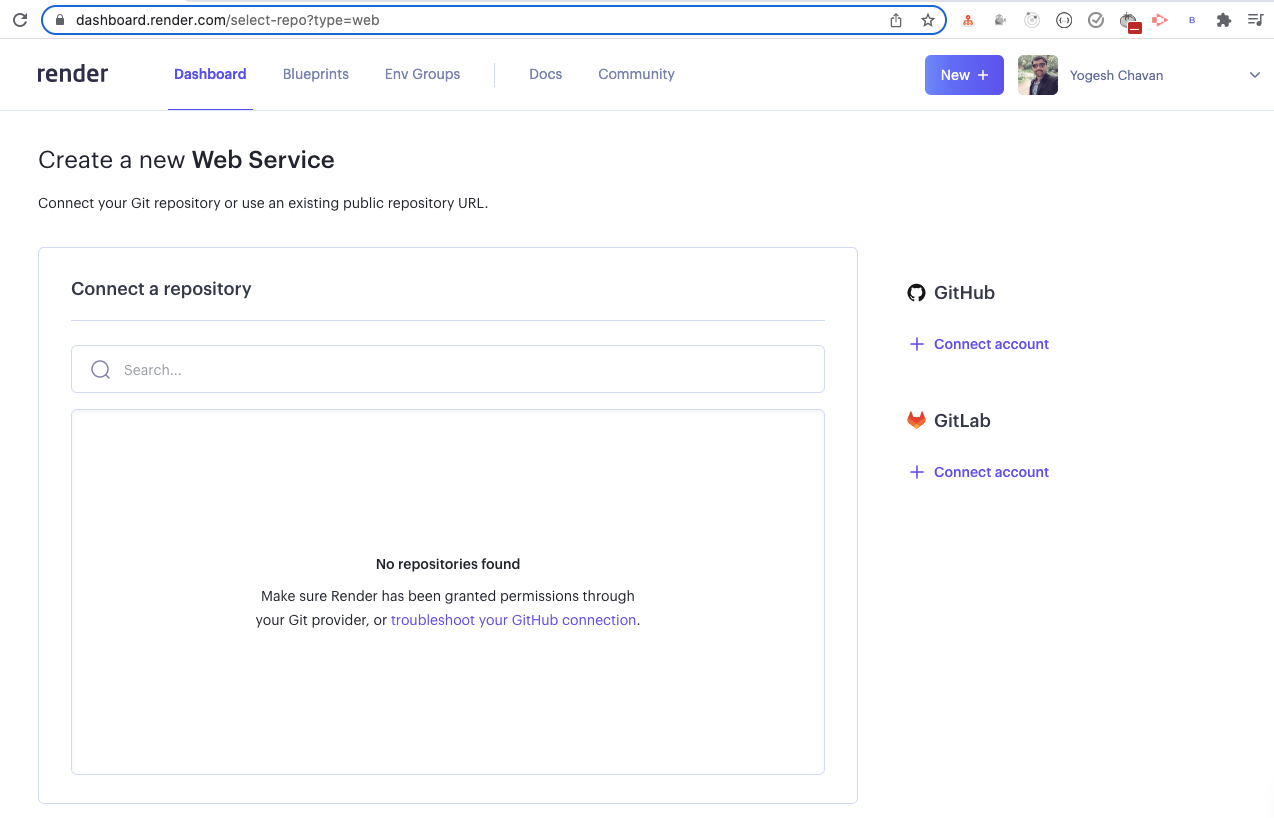
https://www.freecodecamp.org/news/how-to-deploy-nodejs-application-with-render/

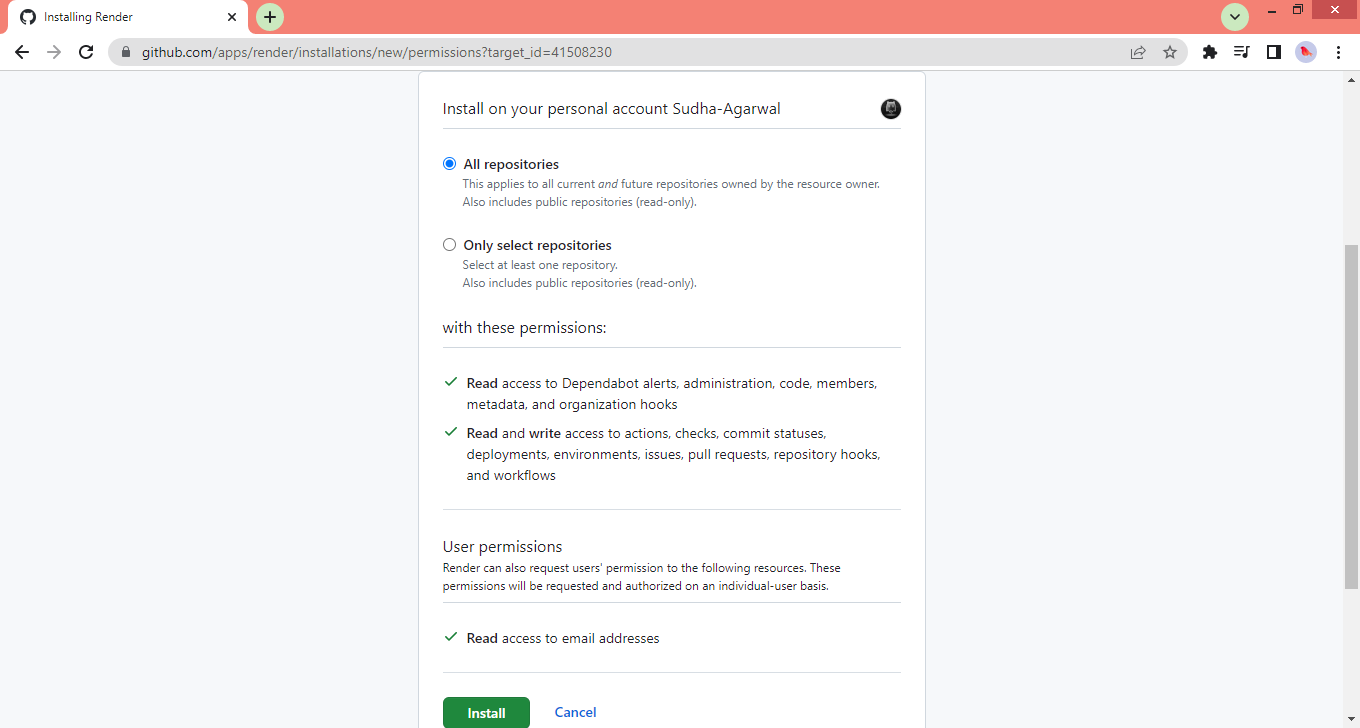


To deploy a Node.js application, click on the New Web Service button under the Web Services option.

You can also click on the New + button displayed in the header just before your profile picture and select Web Service option.

Once clicked, you will see the following screen:





Click on the Configure link and you can give permission to select all your GitHub repositories or only selected repositories.

I like to give access to only selected repositories which I currently need to deploy. So I selected the Only select repositories option.

Next, click on the Select repositories button displayed below the option and select the GitHub repository which you want to deploy.

